8-7-2008

Classifier System Learning of Good Database Schema

Mitsuru Tanaka
University of New Orleans

Classifier System Learning of Good Database Schema

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
The Department of Computer Science

By

Mitsuru Tanaka

B.E., Hiroshima Kokusai Gakuin University, Japan, 2001

August, 2008

Dedication

This thesis is dedicated to my parents Mr. Chiaki Tanaka and Mrs. Hisako Tanaka, and my brothers Yusaku and Shingo.

Acknowledgement

I would like to thank my thesis advisor Dr. Shengru Tu. Dr. Tu introduced me to a research project, which required knowledge of database and artificial intelligence fields and later motivated me to write a thesis on it. Artificial intelligence was a frontier field to me. The fusion of the two fields presented many questions for me to explore in the project. I thank Dr. Tu for answering to my questions and being supportive throughout the completion of my thesis.

I would also like to thank Dr. Adlai DePano, and Dr. Golden Richard, III, for becoming my thesis committee members.

I would also like to thank Dr. Lingyan Shu, who was the team leader in the project, for helping me in the artificial intelligence studies.

I would also like to thank other faculty members and staffs for providing me with a wonderful learning environment.

Table of Contents

List of Figures

List of Tables

Abstract

This thesis presents an implementation of a learning classifier system which learns good database schema. The system is implemented in Java using the NetBeans development environment, which provides a good control for the GUI components. The system contains four components: a user interface, a rule and message system, an apportionment of credit system, and genetic algorithms. The input of the system is a set of simple database schemas and the objective for the classifier system is to keep the good database schemas which are represented by classifiers. The learning classifier system is given some basic knowledge about database concepts or rules. The result showed that the system could decrease the bad schemas and keep the good ones.

Keywords and Phrases: classifier system, machine learning, genetic algorithms, and relational database.

Chapter 1: Introduction

A classifier system is a machine learning system which uses genetic algorithms. Genetic algorithms are based on the principle of Darwin's survival of the fittest. They are derived from a computational model of evolutionary genetics. They also have been applied to search and optimization problems. Genetic algorithm based classifier systems have been applied to a variety of learning tasks. In this thesis, the classifier system is applied to learn good relational database schemas.[1]

1.1 Applying the Classifier System to the Learning of Good Database Schema

Database design involves complex tasks. Databases are widely used in banking, airlines, universities, or credit card transactions. Such databases are designed to manage large bodies of information. To map the information to relational database, there are many steps involved in the design. Database design usually involves the following phases [1]: characterizing the data needs of the prospective database users, conceptual-design, creating a specification of functional requirements, logical design, and physical design. In these phases, there are many concepts, theories, or rules, which help the database designers to produce good database schemas. These five phases involve complex tasks when the information is large.

[1] The ideas, methods, and algorithms of applying classifier systems to learn database schemas were provided by Dr. Lingyan Shu who can be reached at lingyan@conciseinfo.com.

In the first phase to characterize the data needs of the prospective database users, the database designer needs to interact extensively with domain experts and users to carry out this task. The output of this phase is a specification of user requirements. If the client is a bank, there may be a requirement that all the customers must be identified by their customer identification values. In addition, there may be a requirement that a customer must always be associated with a particular banker, who may act as a loan officer or personal banker for that customer. Because of the first requirement, the database designer must set up a constraint that the identification value given to the customer cannot be assigned to other customers. The second requirement raises a question: what if the banker associated with a customer quits? As the information gets complex, the requirements can be large. These requirements or restrictions tend to complicate the designing of databases.

In the conceptual-design phase, the database designer translates the specification of requirements to a conceptual schema of the database. The output is a schema that provides a detailed overview of the enterprise. Usually, the entity-relationship model (E-R model) [2] is used for this purpose. The output schema is described as an entity-relationship diagram with the E-R model.

In the E-R model, a thing or object in the real world that is distinguishable from all other objects is called an entity. For example, each person in an enterprise is an entity. An entity is represented by a set of attributes, which are descriptive properties possessed by each member of an entity set. For example, a person at a bank with an identification number, a name, or a street may be the example attributes. An entity set is a set of entities of the same type that share the same properties, or attributes. Therefore, the all customers with the bank may be defined as a customer entity set. An association among several

entities is defined as a relationship in the E-R model. For example, the relation between a customer named Johnson and his associated banker named Smith can be described as a relationship. A relationship set is a set of such relationships of the same type. The relationships between customer and associated banker entity sets can be described as a relationship set. Even though the E-R model captures such semantics among data and depicts in an E-R diagram, there are some known issues.

Consider the entity set CUSTOMER with its attributes CUSTOMER_ID, CUSTOMER_NAME, CUSTOMER_CITY, and CUSTOMER_STREET in (a) of Figure 1.1. It is possible to argue that (b) of Figure 1.1 can also be constructed from (a). In (b) of Figure 1.1, TELEPHONE is considered to be an entity set with the attributes TELEPHONE_NUMBER and LOCATION. Since TELEPHONE belongs to CUSTOMER, there is a relationship CUST_TELEPHONE. This way, it is possible to define a set of entities and the attributes among them in a number of different ways. Database designers have to choose what is represented as entities and attributes. When attributes are the choice in the decision, there will be a case that a relationship set is constructed as in Figure 1.1. Depending on the choice, the schema will affect the later phases.

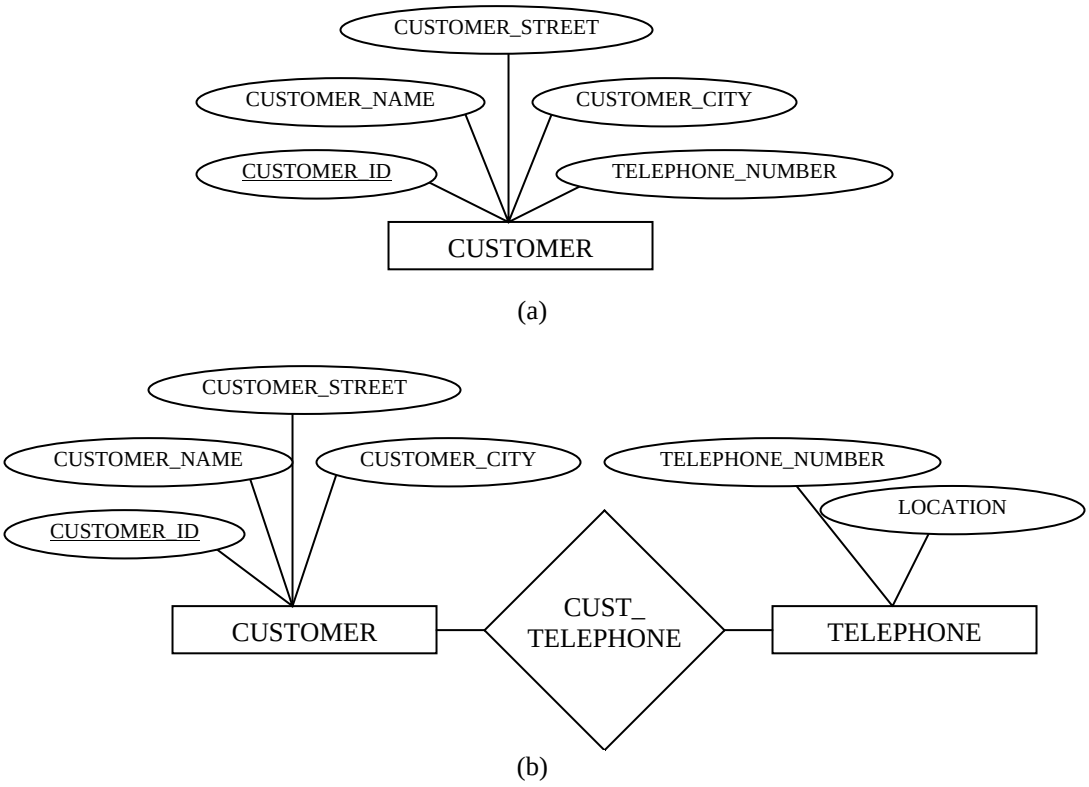

Figure 1.1: Alternatives for CUSTOMER and TELEPHONE

In the specification of functional requirements, users describe the kinds of operations (or transactions) that will be performed on the data. Example operations include modifying or updating data, searching for and retrieving specific data, and deleting data. Users usually have their own business requirements. The schema that comes out of the second phase certainly affects this phase. Suppose that a bank has entities as depicted in Figure 1.2. There are ACCOUNT and LOAN, but there is no CUSTOMER that we saw from the previous example, and the design requires the information about customers that are recorded in ACCOUNT. In the specification of functional requirements, a bank may state some requirements as follows: whenever any type of new customer comes, the banker can record their names and address information

in the database. Suppose a customer comes to a bank, but the customer is not coming to create an account or to make a loan. The banker has to keep the names and address of the customer. Therefore, the banker needs to update the ACCOUNT, since it is the only entity set containing the attributes CUSTOMER_NAME, CUSTOMER_CITY, and CUSTOMER_STREET. The problem is that the customer does not have values for ACCOUNT_NUMBER and BALANCE. Thus, the banker has to update with null values. However, the primary key constraint does not allow the operation, and the update is impossible. This way, the schema that came out of the second phase also needs to meet the functional requirements. The users of database usually have many such requirements, and the design must be done to meet these requirements.

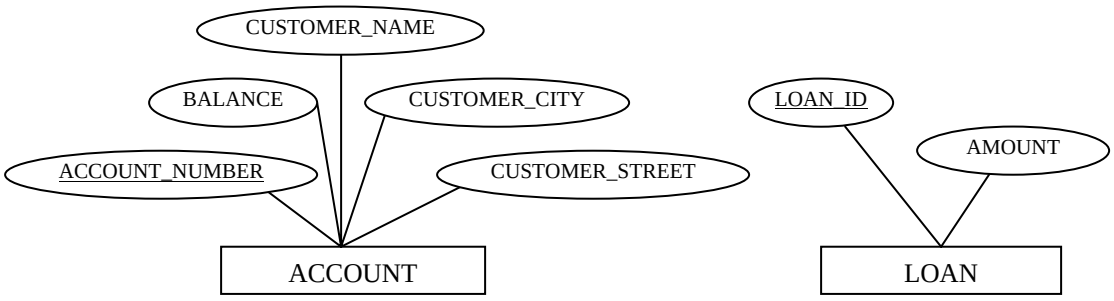

Figure 1.2: An E-R Diagram without CUSTOMER Entity Set

In the logical-design, the fourth phase, the designer maps the conceptual schema onto the implementation data model of the database system that will be used. The implementation data model is typically the relational data model [3]. With the relational data model, the output is the set of relation schemas, which allows us to store information without unnecessary redundancy, yet also allow us to retrieve information easily. This is accomplished by designing schemas that are in an appropriate normal form, and the

process is called normalization [3]. If the E-R diagrams are carefully designed, the relation schemas should not need much further normalization [1]. Normalization can be left to the designer's intuition during E-R modeling, and can be done formally on the relations generated from the E-R model. Poor E-R model produces poor relation schemas. The poor relation schemas have repetition, update, insertion, and deletion anomalies [4]. These anomalies are solved formally in the normalization of relation schemas. There are first, second, third, Boyce-Codd, fourth, and fifth normal forms. These normal forms are based on dependency structures. The BCNF and lower normal forms are based on functional dependencies, fourth normal form is based on multivalued dependencies, and fifth normal form is based on projection-join dependencies.

One of the formal approaches is based on the notion of functional dependencies. Functional dependency (FD) shows the dependency among attributes. For example, LOAN_ID always determines the AMOUNT in Figure 1.2, if a unique value is assigned to each loan at the bank. In this case, the following FD holds:

LOAN_ID $\rightarrow$ AMOUNT.

Here, the FD states that the value of AMOUNT is always determined by the value of the LOAN_ID. During the normalization process, database designers have to deal with the functional dependencies. In the Boyce-Codd normal form (BCNF), there is a need to calculate all the FDs among all the attributes in a relation schema, denoted F^+, by inference rules. This is an expensive computation and falls into a NP-complete problem [5]. These functional dependencies are also used to find keys for relation schemas. Verifying if a given schema has a minimum cardinality key for the relation is also a NP-

complete problem [5]. Database designers have to deal with the functional dependencies, in which such complexities exist.

In the last physical-design phase, database designers specify the physical features for the database schemas that came out of the logical-design phase. Those features are for example, file organization or internal storage structures. These physical structures are carried out and changed in a relatively easy way. On the other hand, logical design is usually harder and changes to logical design will affect a number of factors, such as queries or updates. Therefore, the important phases are the four phases before the physical design.

While nearly all the traditional industries have had their effective database models established, some unconventional information systems in which multivalued dependencies are significant still raise challenging database schema design problems. The goal of our classifier system is to learn the database design tasks. The classifier system tries to classify candidate database schemas to good and bad database schemas, given some simple database knowledge. An anticipated result is to produce reasonable database schema for the emerging information systems in which higher level of normalization is needed.

1.2 Organization

In Chapter 2, background is provided. In Chapter 3, the classifier system is discussed with details. Genetic algorithms and the components of classifier system are stated. Chapter 4 shows the implementation of the system written in Java. How the database schemas are put into the architecture of classifier system is stated. Chapter 5

shows the result of an experiment with a simple database schema. Chapter 6 shows the conclusion.

Chapter 2: Background

In this chapter, I have reviewed the related fields and existing literatures. First, artificial intelligence is reviewed. The definition and the branches of artificial intelligence are provided. In the following section, machine learning is presented. The definition of learning and the example are provided. Finally, two successful applications of classifier system are introduced.

2.1 Artificial Intelligence

Artificial intelligence is a term coined by John McCarthy in 1958 [6]. It is defined as the science and engineering of making intelligent machines, especially intelligent computer programs [7]. McCarthy defines the intelligence as the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines. The intelligence involves mechanisms. If doing a task requires only mechanisms that are well understood today, computer programs can give very impressive performances on these tasks. Such programs can be considered somewhat intelligent.

Many branches of artificial intelligence exist, such as search, pattern recognition, or learning from experience. In search, there is usually a requirement to examine a large number of possibilities. For example, in a chess game, there would be an exponential number of next possible movements. How to search or discover the best among such candidates is the main goal of search. Pattern recognition filters raw data and identifies the category to which the data belongs to. A program makes observations of the

characteristics in objects and compares what it sees with a pattern. In the recent movie *WALL-E*, WALL-E was comparing a spork with two patterns of spoon and fork. The robot's action would require a pattern recognition program. In learning from experience, a machine will try to learn just as humans learn from experience. A checker program may improve its performance by successfully choosing the best move or relatively bad move by evaluating each movement it made.

Many example applications of artificial intelligence exist today, such as speech recognition, understanding natural languages, or heuristic classification. Recently, people often speak with a machine over the phone. When one calls a company such as a telephone company, a machine might answer requesting identification information. One's date of birth might be stated. Then, the person may hear "Sorry, I could not hear that. Please repeat the date of birth." Most likely, the telephone operator is an application of speech recognition, which could not recognize the speech. The application that understands a natural language may understand an event. For example, by scanning a headline from a news company, the machine can understand what has happened. An application of heuristic classification may advise whether to accept or reject a proposed credit card purchase. Given the information about the owner of the credit card's credit history, or the item he is trying to buy, the application makes the decision. There may also be an application that detects credit card fraud.

The scope of this thesis focuses on the branch of learning from experience. The ability to learn is one of the central features of intelligence. A chess machine named Deep Blue defeated a human world chess champion in 1997. With its powerful computational capability and given knowledge, the machine overwhelmed the intelligence of the human

world chess champion for the first time in history [8]. The success story of Deep Blue further strengthened the possibility that a machine may learn just as a human learns. The story brought up implications that artificial intelligence may be applied to various areas, such as molecular dynamics, financial risk assessment, and decision support [9]. The future may witness machines that learn from experience and behave just as humans do in these areas. The branch of learning from experience has a close link to the field called machine learning.

2.2 Machine Learning

Machine learning is a field of study concerning with a question of how to construct computer programs that automatically improve with experience [10]. Since computers were invented, researchers wondered whether computers might be made to learn. The definition of learning is given as follows:

> A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E* [10].

For example, a computer program that learns to play checkers might improve its performance as measured by its ability to win at the class of tasks involving playing checkers games, through experience obtained by playing games. For this example, the task *T* is playing checkers. The performance measure *P* is the percent of games won against opponents. The experience *E* is playing games against itself.

Many algorithms have been invented for learning tasks, and a theoretical understanding of learning has emerged. As a result, many types of machine learning

appeared. One type is called Genetics-Based Machine Learning (GBML). One of the most common GBML architecture is called classifier system, which was introduced by John Holland in 1978 [11]. Classifier system is a machine learning system which uses genetic algorithms and reinforcement learning. Genetic algorithms provide a learning method motivated by an analogy to biological evolution and are successfully applied to a variety of learning tasks and other search or optimization problems. Reinforcement learning uses reward to the actions that the system performs. Good actions are given positive reward values and bad actions are given negative reward values. Classifier system employs this genetic algorithms and reinforcement learning. Since the classifier system is introduced, there have been a lot of applications.

2.3 Previous Works on Classifier System

David Goldberg applied the classifier system to a natural gas pipeline control task in 1978 [12]. Natural gas was provided by a complex system composed of hundreds or thousands of miles of large-diameter pipe consuming thousands of hours of compression horsepower day in and day out. The consumption rate fluctuates depending on the time of year and time of day. If the compressor is always running at its full power, the electricity expenses will be high. Therefore, the control of flow rate must be done efficiently. Also, the pipelines are subject to random leak events. The tasks assigned to the classifier system were to send a flow rate as necessary, and alarm when leak is suspected.

As an input, environmental state was transmitted to the classifier system, such as inlet and outlet pressure, inflow and outflow, upstream pressure rate, time of day, time of year, and temperature. The classifier system was rewarded by a human trainer if it learns

to both operate the pipeline and alarm correctly. As the system takes actions, it learned good pipeline operations. After 400 days of experience with the environment, the classifier system learned good actions and bad actions successfully.

Stewart Wilson applied the classifier system to learning of a Boolean function [13]. The classifier system learned a six-line multiplexer. The multiplexer function is depicted schematically in Figure 2.1. Six signal lines come into the multiplexer. The signals on the first two lines (the address or A-lines) are decoded as an unsigned binary integer. This address value is then used to indicate which of the four remaining signals (on the data or D-lines) is to be passed through to the multiplexer output. In Figure 2.1, the address signal 11 decodes to 3, and the signal on data line 3 (signal = 1) is passed through to the output (output = 1). Initially, the multiplexer function is not known by the classifier system. Every time the input comes to the address line, the system tries to output the correct output. When the output is correct, the system receives an appropriate reward. As the system iterates, the classifier system learned the multiplexer and decoded the Boolean function.

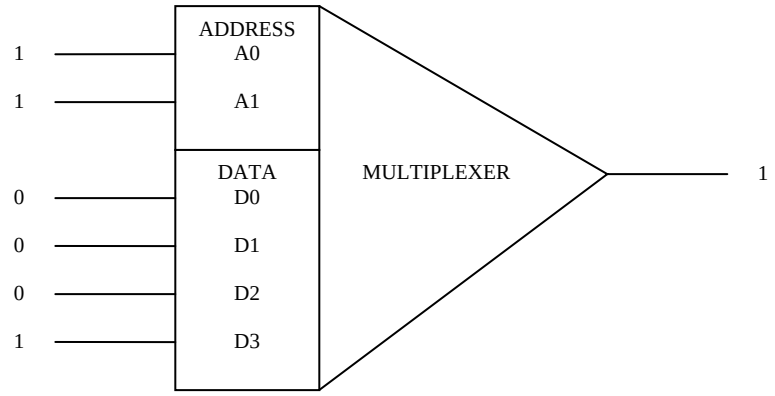

Figure 2.1: A Six-bit Multiplexer Function[2]

These two were some of the most successful applications of classifier system [11]. GBML such as classifier systems are applied to various fields, such as medicine, business, or computer science. Classifier systems are used in arbitrary environment as seen in the Goldberg's application of classifier system to natural gas control. Such a dynamically changing environment involves a certain level of complexity. Classifier system interacts with such an environment with a large number of requirements, and learns to increase its performance by processing many representation of knowledge with its mechanisms. Such classifier system is often applied to fields where a lot of complex tasks are involved.

[2] Figure 2.1 is a reproduced version of a figure from the book "Genetic Algorithms in Search, Optimization & Machine Learning." The granted permission is attached at the Appendix of this thesis.

Chapter 3: An Intensive Study on Classifier System

My thesis project was on applying the classifier system to database design. The focus on my project was on the learning process of the classifier system. A precursor of my project was an intensive study on the classifier system. In this chapter, the details of classifier system are presented.

Classifier system, or learning classifier system, is the most common architecture of genetics-based machine learning systems (GBML). Classifier system and learning classifier system are often used interchangeably. Throughout this thesis, classifier system is used for the term of the architecture. Classifier system learns syntactically simple string rules called classifiers to guide its performance in an arbitrary environment [11]. Classifier system has three components: the rule and message system, the apportionment of credit system, and the genetic algorithms.

3.1 Genetics-Based Machine Learning

The theoretical foundation of GBML was established by John Holland [14]. Holland studied natural adaptive system and its environment. He investigated whether it is possible to formulate some key hypotheses and problems from relevant parts of biology. He outlined how artificial systems can generate procedures enabling them to adjust efficiently to their environments rigorously as natural biological systems. GBML borrows the idea of genetics, the genetic properties of features of an organism. Such GBML uses the genetic search called genetic algorithms, as their primary discovery heuristic.

3.2 Genetic Algorithms

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics [11]. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. Genetic algorithms have two goals. One is to abstract and rigorously explain the adaptive processes of natural systems. The other is to design artificial intelligence software that retains the important mechanics of natural systems. If higher levels of adaptation can be achieved, existing systems can perform their functions longer and better. Features for self-repair, self-guidance, and reproduction are the rule in biological systems. The secrets of adaptation on natural systems can be studied from natural biological systems. Genetic algorithms have a power to adaptively interact with an environment. Genetic algorithms also have robustness.

3.2.1 Robustness of Genetic Algorithms

Genetic algorithms came out with a central theme, robustness. The search is robust if the balance between efficiency and efficacy necessary for survival in many different environments [11]. Traditionally, there were three main types of search methods: calculus-based, enumerative, and random. Genetic algorithms are more robust search method, compared to the three traditional search methods.

Calculus-based methods are subdivided into two classes: indirect and direct. The indirect methods seek local extrema by solving the usually nonlinear set of equations resulting from setting the gradient of the objective function equal to zero. On the other hand, direct methods seek local optima by hopping on the function and moving in a

direction related to the local gradient. For example, the objective may be to find the peak of a mountain as depicted in Figure 3.1. Indirect methods start finding the peak by setting the initial points with slope of zero in all directions. Direct methods climb the function in the steepest direction. Both of them successfully find the peak of the mountain. However, when the mountain has multiple peaks as depicted in Figure 3.2, these methods do not always find the highest peak, since their search relies on the local best in a neighborhood of the current point. The calculus-based methods rely on the derivatives of local and it is hard to find the maximum value always, and thus they are not robust.

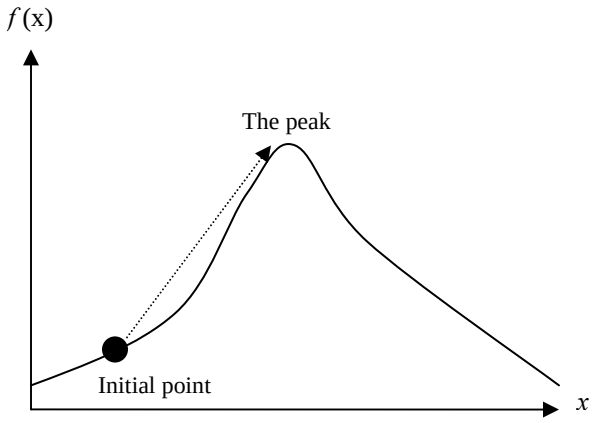

Figure 3.1: A Mountain with a Single Peak

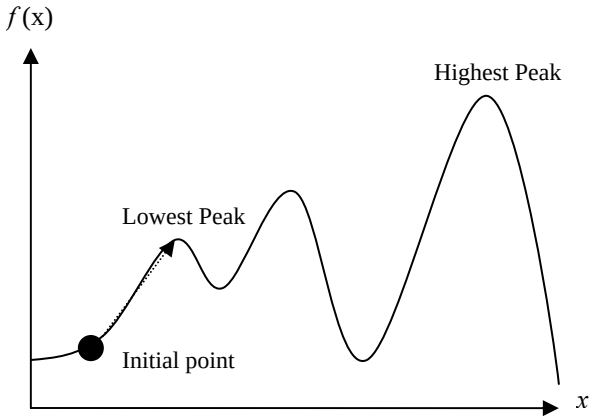

Figure 3.2: A Mountain with Multiple Peaks

Enumerative search method is straightforward. Given a finite search space, it enumerates all the function values at every point, one at a time. The simplicity is very attractive and the approach is very human-like. When the number of possible solutions is small, then this method can be used. However, many practical spaces are too large to search one at a time. In terms of efficiency, the method is not meeting the robustness.

Random search started attracting researchers as the shortcomings of calculus-based and enumerative methods were recognized. In random search, the possible solutions are randomly picked. The search ends when the function value reaches the specified value by a user. Unfortunately, random search method is expected to do no better than enumerative method, and thus lacking in efficiency [11].

These three traditional search methods are not robust search methods. The real world problem spaces are discontinuous, noisy, and vast multimodal. There are many of such functions as depicted in Figure 3.3. With calculus-based search, the result may not be the maxima or the derivative may not be found. Enumerative and random search is not

efficient. Genetic algorithms differ from these methods in the following search procedures:

1. The algorithms work with a coding of the parameter set, not the parameter themselves.
2. The algorithms search from a population of points, not a single point.
3. The algorithms use payoff of objective function information, not derivatives or other auxiliary knowledge.
4. The algorithms use probabilistic transition rules, not deterministic rules.

Genetic algorithms are theoretically and empirically proven to provide robust search in complex spaces [11]. The organization of the algorithms borrows the idea of natural selection described by Charles Darwin, and the operators of the algorithms have a close link to the theory.

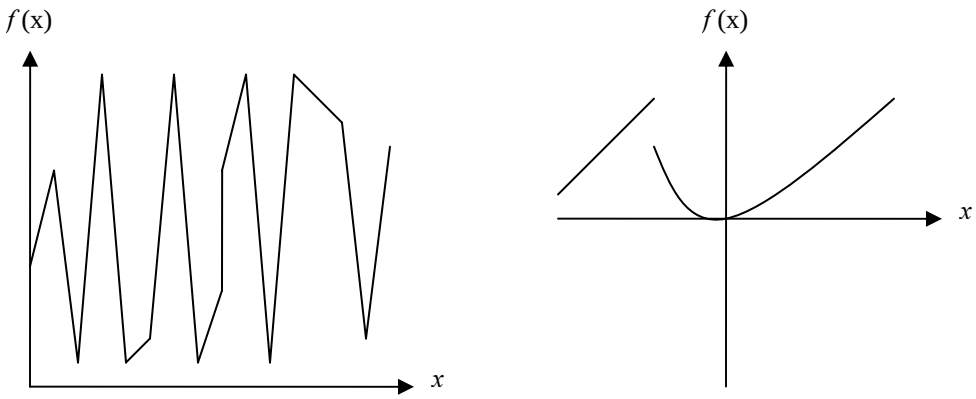

Figure 3.3: Noisy and Discontinuous Functions

3.2.2 Natural Selection

Darwin wondered about the origin of species and the evolution of organic living things. He wondered how species differ from each other more than do the species of the

same genus. He thought that all these results follow inevitably from the struggle for life. Darwin introduced the term, natural selection [15], in 1859 and theorized about the evolution. Natural selection is a principle that states as follows: “Owing to this struggle for life, any variation, however slight and from whatever cause proceeding, if it be in any degree profitable to an individual of any species, in its infinitely complex relations to other organic beings and to external nature, will tend to the preservation of that individual, and will generally be inherited by its offspring. The offspring, also, will thus have a better chance of surviving, for, of the many individuals of any species which are periodically born, but a small number can survive.” Living things reproduce. Organic living things are subject to a strict natural environment. Living things that have adapted to the environment have a higher chance to survive. Living things mutate themselves to survive the environment, and the mutation is inherited by the reproduced successors. Genetic algorithms have operators which are artificial versions of natural selection: reproduction, crossover, and mutation.

3.2.3 Reproduction

Reproduction in genetic algorithms is a process in which individual strings are copied according to their objective function values, f. Biologists call this function the fitness function. The value indicates how much the individual is fitting to the environment for survival. We can think the function f as the profitability or goodness that we want to maximize. As stated in Section 3.1, genetic algorithms work with a coding of the parameter set and search from a population of points. The parameter set is coded as a finite-length string over some finite alphabet.

Let's consider an example population having four strings of five-digit binary numbers as depicted in Table 3.1. The objective function value is as follows:

$f(x) = x^2$, where $0 \le x \le 31$.

For example, in row 4 of the population, the value of f is $19^2 = 361$ with the objective function equation. Other strings also have the fitness values according to the objective function. In reproduction, genetic algorithms pick some strings to generate offspring for the next generation according to their fitness function values. Then the reproduction process replaces the strings having worst fitness values with the newly created strings from the picked ones. Thus, the members who have higher fitness values tend to survive; those members are reproduced and inherited by the offspring.

No.	String	Fitness	% of Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0

Table 3.1: Example of Four Strings[3]

The percentage of the population total fitness is also shown in Table 3.1. In genetic algorithms, a weighted roulette wheel by the percentage is used. Figure 3.4 shows such a roulette wheel. The roulette wheel is a tool used to decide who to select to generate new strings of offspring. To pick the offspring for the next generation, the roulette wheel is turned. In this weighted wheel, the string No.1 has 14.4 percent of

[3] Table 3.1 is a reproduced version of a figure from the book "Genetic Algorithms in Search, Optimization & Machine Learning." The granted permission is attached at the Appendix of this thesis.

probability to be picked for the next generation. The string No.2 has 49.2 percent of probability, and thus has the highest probability to be picked for the next generation. In this way, more highly fit strings have a higher number of offspring in the succeeding generation. During the reproduction, picked strings are mated at random, and then a crossover may proceed on each pair.

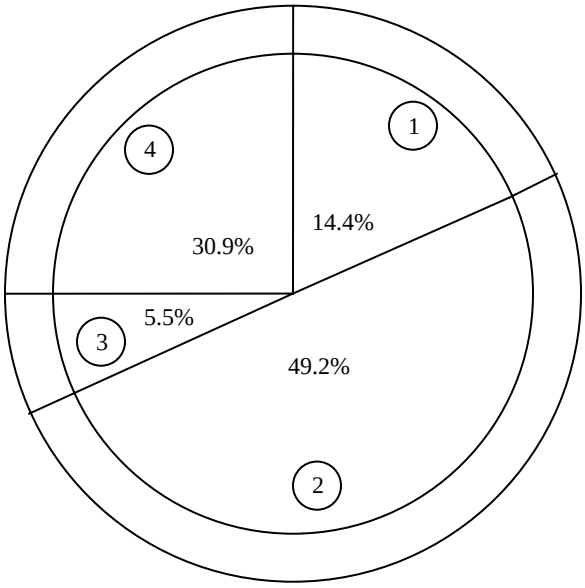

Figure 3.4: Roulette Wheel for Reproduction[4]

3.2.4 Crossover

The crossover operation is done as follows: an integer position k along the string is selected uniformly at random between one and the string length less one [1, $l - 1$]. Two new strings are created by swapping all characters between the positions $k + 1$ and the length l inclusively. For example, consider the following mated strings A_1 and A_2 with a value 4 for the k:

[4] Figure 3.4 is a reproduced version of a figure from the book "Genetic Algorithms in Search, Optimization & Machine Learning." The granted permission is attached at the Appendix of this thesis.

A_1 = 0 1 1 0 | 1

A_2 = 1 1 0 0 | 0

The character | is a separator and put at the position dividing the string. Here, the swapping happens at the 5th character for each string. The resulted strings are as follows:

A'_1 = 0 1 1 0 0

A'_2 = 1 1 0 0 1

In these offspring, the characters before the position 4 are not swapped. The last element of the string is modified by the swapping. The 0 of the A_2 replaced the 1 of A_1. The 1 of A_1 replaced the 0 of A_2. Crossover happens in this way on each pair of the mated strings.

Each string has notions of what is important to be fitted in the environment. The pairs picked during the reproduction are believed to have high-quality such notions. For the binary example, the strings with high-quality notions are the ones who have 1 at the first element. According to the objective function, to be fitted, the strings have to maximize the fitness values by having 1s at lower positions, since the exponent increases as the position goes to the left in the strings of binary representation. Such strings are considered to be fitted to this example environment. The crossover operation performs the exchange of high-quality notions between relatively fitted strings. Just as humans borrow ideas that worked well in the past, crossover operation trades such information between strings. When we notice a new idea, our performance sometimes gets better. Crossover is an operation of such an information exchange. The strings with low fitness values may get a chance to have new notions by exchanging information with other relatively fitted strings during the crossover operation. Following the reproduction and crossover operations, there is another operator called mutation.

3.2.5 Mutation

Mutation is the occasional random alteration of the value of a string position. Suppose there is the following string *A*:

A = 0 0 1 0 1

Mutation randomly picks the position. Let's suppose the first element of the string is mutated. The alteration is between 0 and 1 in the binary example. Since the *A* has 0 at the first element, the value is altered to 1 as follows:

A' = 1 0 1 0 1

Mutation prevents from losing some potentially useful genetic material, such as the notion that lower positions with value 1 increases the fitness value. Consider an example of Table 3.2. There are four strings again, but each string has 0s for all the elements, and thus all have the fitness value 0. In this case, reproduction and crossover do not have any effect. Mutation takes place so that some strings will have new useful notion.

No.	String	Fitness	% of Total
1	00000	0	0
2	00000	0	0
3	00000	0	0
4	00000	0	0
Total		0	0

Table 3.2: Four Strings with 0 Fitness Value

Suppose that No.1 and No.2 are picked and mated during the reproduction and crossover. They do not affect in any way. Thus, we have two newly generated offspring that have 0s in all the positions. Replacing these offspring with some of the members has

no effect. Suppose a mutation happens on the offspring, at the first element of the strings, then we get the following:

A_1' = 1 0 0 0 0

A_2' = 1 0 0 0 0

Then, let's replace these with strings No.3 and No.4. The table change as in Table 3.3 with the new offspring. The mutation operation provides a means to inject a new notion to the strings. Therefore, the operator is useful when no strings have good notions to increase the performance in the environment.

These are the operators that genetic algorithms perform. These three operators are computationally powerful and applied to many search problems. Within the strings that they process, there are always similarities. Strings that have relatively high fitness values share their particular similarities; strings that have relatively low fitness values also share their particular similarities. Genetic algorithms see these similarities in the strings and those similarities help genetic algorithm to search. The similarity is called similarity templates or schemata.

No.	String	Fitness	% of Total
1	00000	0	0
2	00000	0	0
3	10000	256	50.0
4	10000	256	50.0
Total		512	100.0

Table 3.3: The Effect of Mutation

3.2.6 Similarity Templates (Schemata)

Genetic algorithms are advised by the similarity templates, or schemata. A schema [16] is a similarity template describing a subset of strings with similarities at certain string positions. Let's consider strings over the ternary alphabet {0, 1, *}. The character * is a "don't care" symbol. A schema matches a particular string if at every location in a schema a 1 matches a 1 in the string, a 0 matches a 0, or a * matches either. For example, if there are five strings {10000, 00000, 11100, 00001, 01110}, a schema *0000 matches two strings {10000, 00000}. *11** will match {11100, 01110}. Sometimes, genetic algorithms are more interested in these similarity templates, rather than the strings themselves. This is because these similarities describe the traits of highly fitted or badly fitted strings. These schemata of short-defining-length are propagated generation to generation by giving exponentially increasing samples to the observed best. This is formally stated in what is called schema theorem.

3.2.7 Schema Theorem (Fundamental Theorem of Genetic Algorithms)

Schema theorem states that short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations [11]. A schema has two properties: schema order and defining length. The order, denoted by $o(H)$, is the number of fixed positions. The defining length of a schema H, denoted by $\delta(H)$, is the distance between the first and last specific string position. For example, for a schema 011**1*, the order is 4 and the defining length is 5. For a schema 0******, the order is 1 and the defining length is 0. Schemata and their properties are interesting notational devices for rigorously discussing and classifying string similarities. They provide the means to analyze the net effect of genetic operators.

In the reproduction, the schema theorem states that, a particular schema grows as the ratio of the average fitness of the schema to the average fitness of the population. Suppose that at a time *t*, there are *m* examples of a particular schema *H* contained in the population *A*(*t*). Then *m* is written as follows:

$$m = m(H,t) .$$

A string A_i gets selected with the following probability p_i:

$$p_i = f_i / \sum f_j .$$

For each representative of *H*, there will be $n * p_i$ copies in the population *A* of size *n*. Therefore, at time *t* +1, there is the following number of examples of a schema *H*:

$m(H,t+1) = n \cdot (f_1 + f_2 + ... + f_k) / \sum f_j$, where 1…*k* are the representatives of *H*.

The equation can also be written as follows:

$$m(H,t+1) = \frac{n \cdot m(H,t) \cdot (f_1 + f_2 + ... + f_k) / m(H,t)}{\sum f_j} .$$

Thus:

$m(H,t+1) = n \cdot m(H,t) \cdot f(H,t) / \sum f_j$, where $f(H,t)$ is the average fitness of the strings representing schema *H* at time *t*. The average fitness of the entire population is written as follows:

$$\overline{f} = \sum f_j / n .$$

Therefore, the *m* at the time *t* + 1 can be written as follows:

$$m(H,t+1) = m(H,t) \frac{f(H)}{\overline{f}} .$$

This equation states that schemata with fitness values above the population average will receive an increasing number of samples in the next generation, while schemata with

fitness values below the population average will receive a decreasing number of samples. During the reproduction, we have this equation as a property.

Crossover disrupts schema sometimes when the separating point is in the defining length of a schema. Thus we have to deal with the probability that a particular schema survives during the crossover. Crossover survival probability is defined as follows:

$$p_s \geq 1 - \frac{\delta(H)}{l-1}.$$

This is because a schema tends to be disrupted whenever a site within the defining length is selected from the length $l - 1$ possible position. If crossover is itself performed by random choice, for example, with probability p_c at a particular mating, the survival probability is in the expression:

$$p_s \geq 1 - p_c \cdot \frac{\delta(H)}{l-1}.$$

Therefore, with reproduction and crossover, considering they are independent, the following equation is given:

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{\overline{f}} \left[1 - p_c \cdot \frac{\delta(H)}{l-1} \right].$$

With reproduction and crossover, schema H grows or decay depending on whether the schema is above or below the population average and whether the schema has relatively short or long defining length.

The mutation operator is the random alteration of a single position with a probability p_m, thus a single position survives at the probability $1 - p_m$. In order for a schema H to survive, all of the order of schema have to survive. Therefore, the probability of surviving mutation is given as follows:

$(1-p_m)^{o(H)}$.

For small values of p_m ($p_m << 1$), the schema survival probability is approximated by the following expression:

$1-o(H)p_m$.

The three operators of genetic algorithms are quantitatively defined as above. The concluded equations are given as follows:

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\overline{f}}\left[1-p_c \cdot \frac{\delta(H)}{l-1}-o(H)p_m\right].$$

A particular schema *H* receives an expected number of copies in the next generation under reproduction, crossover, and mutation as given in this equation. Also, short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations. This conclusion is defined as schema theorem[5], or fundamental theorem of genetic algorithms. The following section shows an example of genetic algorithm operations for a knapsack problem, where some schemata are processed as concluded in the schema theorem.

3.2.8 A Simple Genetic Algorithm for a Knapsack Problem

Knapsack problem is one of NP-hard problems [17]. Genetic algorithms are often applied to this kind of complex problems to search the space. Knapsack problem is defined as follows:

We are given an instance of the knapsack problem with item set *N*, consisting of *n* items *j* with profit p_j and weight w_j, and the capacity value *c*. Then the objective is

[5] The schema theorem stated here is from the book "Genetic Algorithms in Search, Optimization & Machine Learning". The granted permission is attached at the Appendix of this thesis.

to select a subset of N such that the total profit of the selected items is maximized and the total weight does not exceed c.

The problem is formulated as follows:

$$\text{maximize } \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to } \sum_{j=1}^{n} w_j x_j \le c,$$

$$x_j \in \{0,1\}, j = 1,...,n.$$

In this section, the genetic algorithms deal with the following particular instance:

There are a water bottle, a pan, a ring, a camcorder, and a laptop. These are single items, and thus the genetic algorithm can not choose to put more than one item of the same item in the knapsack. The profit and weight of each item is given in Table 3.4. The capacity limit is 50. The genetic algorithm has to not only maximize the total profit, but also choose items carefully so that total weight of the items does not exceed the capacity limit.

	Water Bottle	Pan	Ring	Camcorder	Laptop
Weight	5	30	1	10	20
Profit	1	2	40	10	20

Table 3.4: Weight and Profit of the Items for a Knapsack Problem

The genetic algorithm has to deal with 128 combinations, in addition to the capacity limit. The combinations can be represented in binary. The example is in Figure 3.5. Here, the three items (Water Bottle, Pan, and Camcorder) are chosen to be put into the knapsack.

The string is 11010 and the total weight and the total profit value are calculated as follows:

$$11010 = 1*5 + 1*30 + 0*1 + 1*10 + 0*20 = 45 \quad \text{(Total Weight)}$$

$$11010 = 1*1 + 1*2 + 0*40 + 1*10 + 0*20 = 13 \quad \text{(Total Profit)}$$

In this solution, the total weight is within the capacity, but the profit could have been better. The example genetic algorithm works with this binary representation of the solution space.

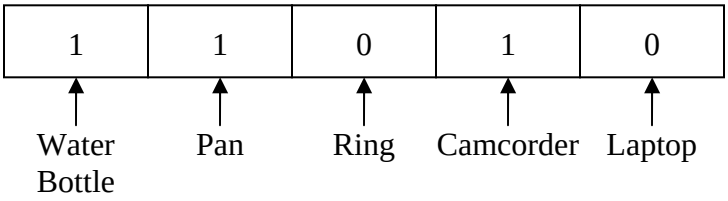

Figure 3.5: An Example Binary Representation of the Strings in a Knapsack Problem

Table 3.5 shows the processing of strings and schemata together by the genetic algorithm for two generations. There are four strings which are randomly generated in the initial population. They are the first generation of the population. The total weight and profit for each string are calculated by the functions in the table. The total weight of the string No.1 is exceeding the capacity limit of 50. Such strings are given a value 0 for the profit.

String Processing						
String No.	Initial Population (Randomly Generated)	Total weight w $\sum_{j=1}^{n} w_j x_j$	Total Profit f $\sum_{j=1}^{n} p_j x_j$	Probability that ith String is Selected $\frac{f_i}{\sum f}$	Expected count (from Roulette Wheel) $\frac{f_i}{\bar{f}}$	Actual count (from Roulette Wheel)
1	0 1 1 0 1	51	0	0	0	0
2	1 1 0 0 0	35	3	0.03	0.14	1
3	0 0 1 1 0	11	50	0.60	2.38	2
4	1 0 0 1 1	35	31	0.37	1.48	1
Sum			84	1.00	4.00	4.0
Average			21	0.25	1.00	1.0
Max			50	0.60	2.38	2.0

Schema Processing			
		Before Reproduction	
		String Representatives	Schema Average Fitness $f(H)$
H_1	1 1 0 * 0	2	3
H_2	* 0 1 * *	3	50

Table 3.5: String and Schema Processing by Genetic Algorithms for a Knapsack Problem

String Processing				
Mating Pool After Reproduction (Cross Site At the Symbol "|")	Mate (Randomly Selected)	Crossover Site (Randomly Selected)	New Population	Total Profit f $\sum_{j=1}^{n} p_j x_j$
0 0 1 | 1 0	2	3	0 0 1 0 0	40
1 1 0 | 0 0	1	3	1 1 0 1 0	13
0 0 1 1 | 0	4	4	0 0 1 1 1	70
1 0 | 0 1 1	3	2	1 0 1 1 0	51
Sum				174
Average				43.5
Max				70

Schema Processing					
After Reproduction			After All Operators		
Expected Count of Strings of a particular Schema	Actual Count	String Representatives	Expected Count of Strings of a particular Schema	Actual Count	String Represen-tatives
0.14	1	2	0.09	1	2
2.38	2	1,3	2.98	3	1,3,4

Table 3.5 (Continued)

The result of the roulette wheel matters in the reproduction. Out of four turns, the string No.3 is picked twice and chosen to be the offspring. The offspring replaces the worst string No.1, and resulting strings are as in the column of the mating pool. Those are the result of reproduction. The crossover takes place on these. The mate and separating point are determined by random choice. The result of the crossover produces the new population. As seen in the table, the sum of the profit is increased.

What would be the bad or high-quality notion? The bad strings would be the ones that have pan, since the pan is relatively heavy and not profitable. Clearly, the ring gives high profit. Therefore, the strings having 1 at the third position would be good ones. Example schemata having such notions are given as H_1 and H_2 in the table. H_1 is a schema having the bad notion, and the H_2 is the schema having the good notion. The H_2 particularly increases according to the schema theorem. In the initial population, No.3 is the string representing the schema. Therefore the value of $f(H_2)$ is 50. According to the schema theorem, we can expect to have $m \cdot f(H) / \overline{f}$ copies of the schema during the reproduction. Therefore, the m for H_2 at time $t+1$ can be calculated as follows:

$$m(H_2, t+1) = 1 \cdot 50 / 21 = 2.38 .$$

In Table 3.5, two strings No.1 and No.3 of the schema in the population after reproduction can be seen. The average of the fitness values of the strings in the mating pool is 33.5. Therefore, we can calculate the m as follows:

$$m(H_2, t+1) = 2 \cdot 50 / 33.5 = 2.98 .$$

After the crossover operation, the number of strings representing the H_2 is 3 (No.1, No.3, and No.4). Whereas the H_2 increases its number of strings, H_1 does not. The expected number of strings representing H_1 is likely to decrease as new population is generated.

This section showed the example of genetic algorithms with a particular instance of knapsack problem. The genetic algorithms usually work with a large number of strings representing the solution space. Some strings representing high-quality notions are expected to increase. By repeating the generation of offspring, the genetic algorithms give the solutions in the form of strings, which get better and better increasing the fitness values of each. Genetic algorithms are computationally powerful and provide means to search. Such genetic algorithms are used as a component in classifier system, where the strings are called classifiers.

3.3 Components of Classifier System

A Classifier system is a machine learning system that learns syntactically simple string rules, called classifiers [11]. The classifiers guide the performance of classifier system in an arbitrary environment. A classifier system consists of three main components:

1. Rule and message system
2. Apportionment of credit system
3. Genetic algorithm

A classifier system is depicted in Figure 3.6. The information of environment comes to the system through detectors. The information is decoded as message by the detectors. Classifiers react to the environmental message. Classifiers are the thought of the system about the environment. Based on the classifiers, the system takes actions to environment through the effectors. When the action of the system is good, the environment gives payoff to the system. Payoff is the incentive for the system to learn. This helps the system

to understand what is good or bad by receiving such a reward. This method is called reinforcement learning [18]. Through these components, classifiers, and reinforcement, the system learns the environment. The objective for the system is to learn the environment and improve the actions. The information flows from the detectors to effectors. The rule and message system helps the flow in the classifier system.

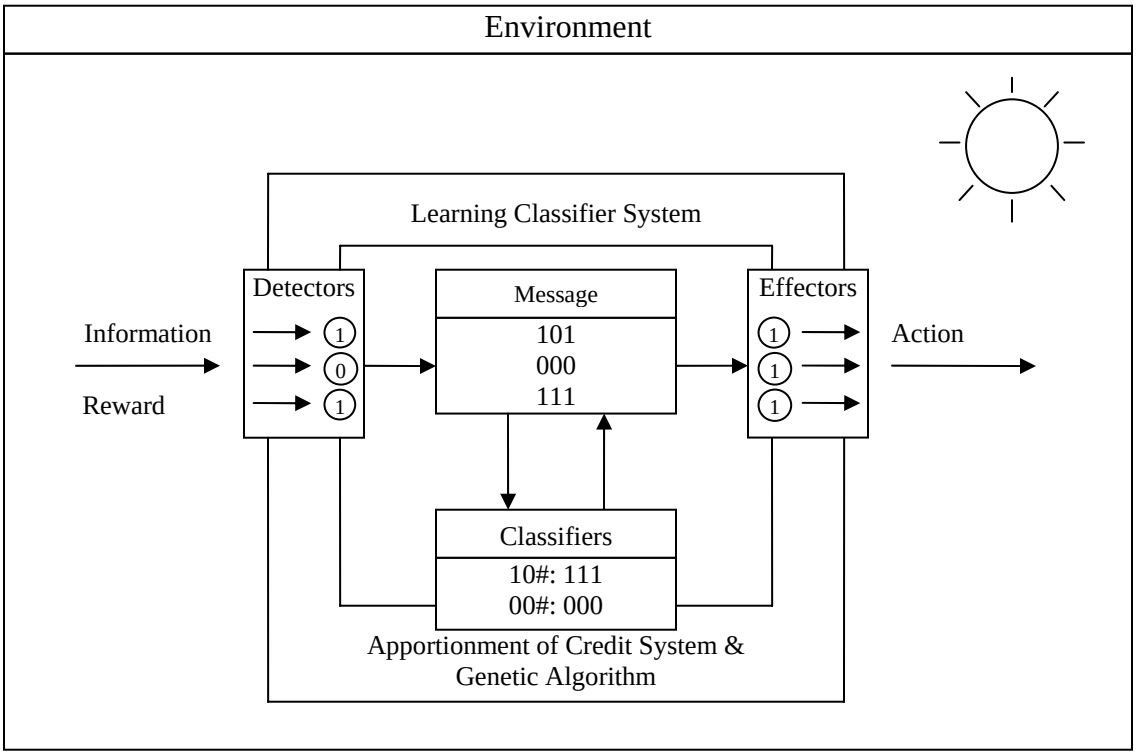

Figure 3.6: A Learning Classifier System Interacting with its Environment[6]

3.3.1 Rule and Message System

The rule and message system of a classifier system is a special kind of production system [19]. The production system has the following form:

if <condition> then <action>.

[6] Figure 3.6 is a reproduced version of a figure from the book "Genetic Algorithms in Search, Optimization & Machine Learning." The granted permission is attached at the Appendix of this thesis.

When the condition is satisfied, the action is fired in a typical production system. The rule and message system employs this form, which is powerful and convenient. The message and classifiers are structurally friendly with this form. If limited to a binary alphabet, the message has the following syntax:

<message> ::= $\{0, 1\}^{l}$.

The *l* indicates the length. For example, a message of a length 3 would be a 110. The message is used as the environmental message causing the system to react, and a part of classifiers.

3.3.2 Classifiers

Classifiers are syntactically simple string rules. The classifier has the following syntax:

<classifier> ::= <condition> : <message>.

The condition is a simple pattern recognition device where a wild card character (#) is added to the underlying alphabet:

<condition> ::= $\{0, 1, \#\}^{l}$.

When the condition is matched by an environmental message, the classifier reacts. For example, the four-position condition #01# matches an environmental message 0010, but it does not match an environmental message 0000. When the classifier's condition is matched, the classifier becomes a candidate to send its message for the next step. When multiple conditions are matched to the environmental message, one classifier is picked from them. The choice is done by the apportionment of credit system, called bucket brigade.

3.3.3 Apportionment of Credit System: Bucket Brigade

Classifiers are ranked by its strength. The strength is determined by the reward point that classifiers acquired from the environment. When multiple classifiers are the candidates to post the message for the next step, the rank by their strength has an effect to the pick. The bucket brigade has two components: an auction and a clearinghouse. When the multiple classifiers are matching, then auction takes place for them. A fish market may be a good analogy for the auction. Buyers from restaurants are the classifiers while the fish is the environmental message. The strength may be the money that the buyers have. The buyers may be interested in particular fish. The salespeople or fishermen sell the fish one by one during auctions. They name a fish, and some of the buyers react to it. Some buyers may raise their hand and tell the seller the bidding price. Thus, they have to compete. At the end, the highest bidder wins and the fish is taken by the winner. Classifiers also compete with other classifiers who matched to the environmental message. The clearinghouse manages the payment keeping a record of those who bid and those who must pay.

3.3.4 Genetic Algorithms in the Classifier System

Genetic algorithms take a role to inject new, possibly better classifiers into the classifier system. Every time an environmental message comes to the system, there is single winner classifier from the population and it takes an action to the environment. Winner classifiers receive reward value from an environment when their actions are correct. The classifier system learns if the performance increases as it experiences the environment. Genetic algorithms replace some of the classifiers to increase the performance of classifier system entirely to learn the environment. The genetic

algorithms reproduce stronger classifiers, and exchange information between classifiers, and sometimes inject new notions.

Chapter 4: Implementation of a Classifier System in Java

I implemented a classifier system in Java programming language. The system consists of about 30 Java classes. NetBeans is chosen as the IDE, since it provides a good control for GUI components in the development processes. Some class diagrams presented in this chapter are generated by NetBeans. In this chapter, some design and implementation features of the classifier system are presented: requirements, input and output, data structures, class diagrams, knowledge given to the system and reward functions, and some source codes with actual operations over the classifiers. How database schemas can be represented as classifiers, and how those classifiers are processed in the framework of classifier system are shown.

4.1 Requirements

The first requirement is to implement the classifier system, which can represent database schemas as its classifiers, and process those in the architecture. The classifier system has powerful mechanisms to process classifier strings with robust genetic algorithms as the search method. Classifiers of database schemas react to environmental messages. Then, the matching schemas compete in an auction. The winner schema is evaluated and receives a reward value by the reinforcement mechanism. Enabling the processes of the classifiers in those mechanisms is the first requirement that the system should be able to do.

Following the implementation, the learning is the main requirement for the classifier system. Classifiers represent database schemas of which some may be good and

some may be bad. The classifier system is given some basic knowledge about database design so that it will think what is good and what is bad. For the learning, the task *T* is to pick the better one among many candidates. The performance *P* is determined by whether the choice gets better as it experiences the environment *E*, which is doing database design by itself. When the system stops, the system prints the strongest classifiers of each database schema for the entity set. Every time the system experiences the database design, it ranks the classifiers by the strength. The classifier system increases the strength of good classifiers and decreases the strength of bad classifiers according to the knowledge.

4.2 Input and Output

Figure 4.1 shows an example input. The classifier system takes a conceptual schema, which is a set of entity sets with the primary-key and possible non-primary key attributes. The classifier system will randomly generate hundreds or thousands of combinations of the attributes for each entity set. Those are represented as classifiers and are the candidates for the final schemas. In addition, the system takes mapping cardinalities between the entity sets. This mapping cardinality shows relationship sets and the cardinality, such as one-to-one or many-to-many. The relationship sets are limited to binary relationships.

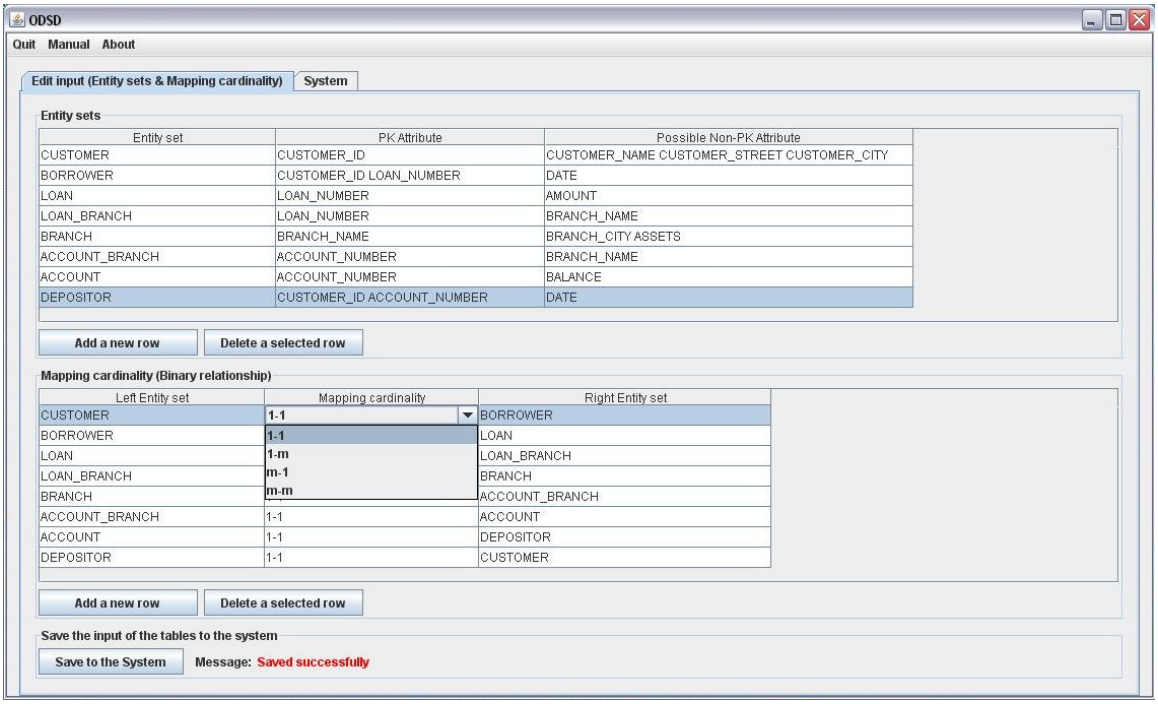

Figure 4.1: The Input to the Classifier System

Figure 4.2 shows the system tab with a pane showing the current input, a result pane, and a plot pane to analyze the learning. The result pane is to print the strongest classifiers for each entity set when the system completes. The strongest classifiers are the database schemas chosen as the best by the system when it stops. The plot is to count the number of classifiers with certain strength values at a certain iteration time point. With the plots, the increase and decrease of the strength can be seen. These are the output of the classifier system.

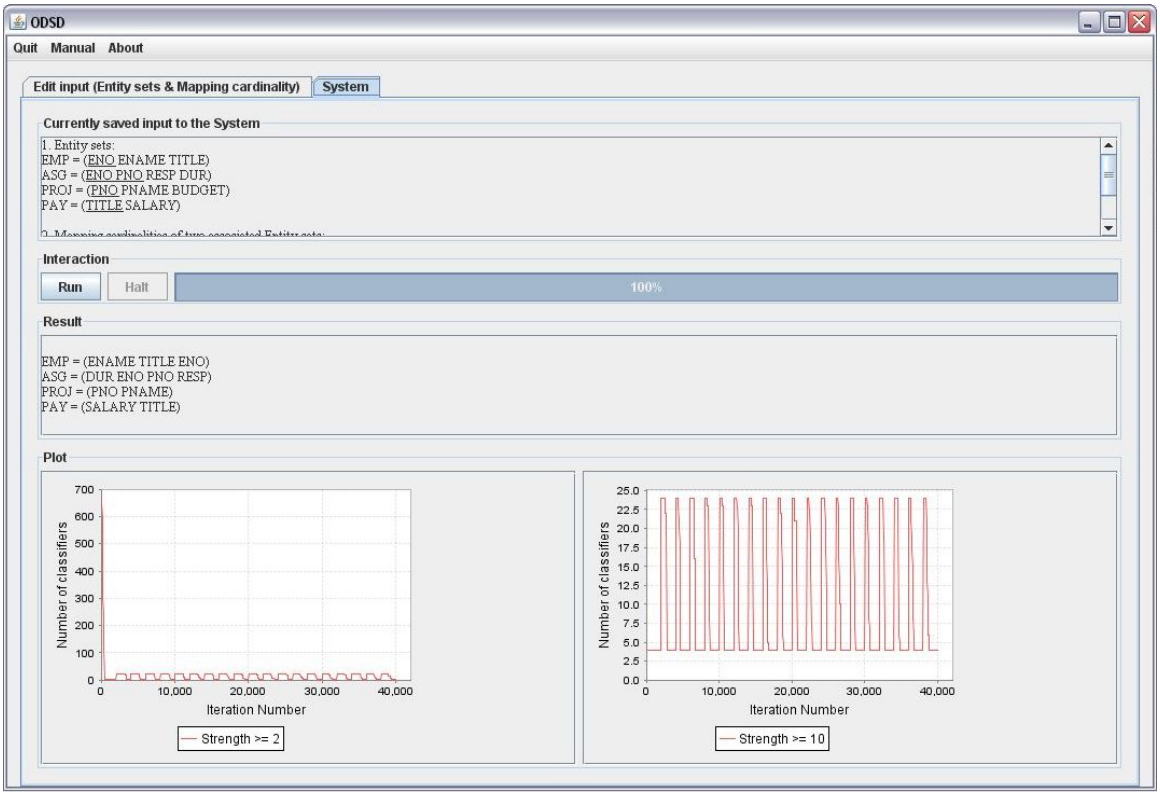

Figure 4.2: System Tab with the Output

4.3 Data Structures

In this section, some important data structures such as classifiers, environmental message, and population are introduced with the syntaxes. Classifiers are the main data structure, which are involved in almost all the operations in the classifier system. Environmental message triggers the operations of classifier system. Population maintains the classifiers and the statistical information.

4.3.1 Classifier

Classifiers represent database schemas. The classifier system randomly generates a large number of such classifiers for every entity set, and all of those are the candidates and have a chance to be printed at the result pane when the system stops. The classifier is named as *ClassType* in the program as in Figure 4.3. Each classifier is programmatically structured this way. The classifier has its condition and message part (action) as in the syntax of classifier. The *strength* is the value indicating how well the classifier is doing in the environment. The *bid* is the value that classifiers bids during the auction. The value of *matchflag* indicates if the instance of the classifier is matching to the environmental message.

```
public class ClassType {

    //component:
    private Condition c; //condition part of this classifier
    private Action a; //action (message) part of this classifier
    private double strength,  bid; //current strength and bidding value of this classifier
    private double ebid; //this classifier's bid plus noise
    private boolean matchflag; //true if this classifier mathes to coming message
```

Figure 4.3: The Declarations of *ClassType*

4.3.2 Environmental Message

Environmental message is the same as the condition part of the classifier. Thus the environmental message has the following syntax:

<environmental message> ::= <condition>.

The environment gives the value of the condition of a classifier, and some classifiers with the same condition will be matched. Therefore, some candidate classifiers with the same

condition react to the environmental message and compete. Population is another data structure maintaining classifiers and the record for who is matching or not matching.

4.3.3 Population

Population is represented as a *PopType* class in the program as in Figure 4.4. The class maintains all the classifiers in a class named *ClassArray*. The *nclassifier* is the size of classifiers. The variables from *pgeneral* to *ebid2* are used in the process of the auction and genetic algorithms. The *initialstrength* is the strength values given to all the classifiers when they are initially generated. The rest of the variables are to maintain the statistical information, such as the max strength value among classifiers.

```
public class PopType {

    //component:
    private ClassArray classifiers;
    private int nclassifier; //number of classifiers populated
    private double pgeneral; //probability for general purpose
    private double cbid; //coefficient bid
    private double bidsigma; //noise deviation
    private double bidtax; //tax rate to classifiers who bid for a auction
    private double lifetax; //tax rate for exitence
    private double bid1; //parameter to calculate a bid
    private double bid2; //parameter to calculate a bid
    private double ebid1; //parameter to calculate a ebid
    private double ebid2; //parameter to calculate a ebid
    private double initialStrength; //initial strength for classifiers
    private double sumstrength; //sum of classifiers' strength
    private double maxstrength;
    private double avgstrength;
    private double minstrength;
```

Figure 4.4: The Declarations of *PopType*

4.4 Class Diagrams

In this section, class diagrams for some important components are provided: rule and message system, apportionment of credit system, genetic algorithms, and reinforcement system. Every time the classifier system receives an environmental message, the classifier performs the same set of operations from these components and repeats the same thing again and again. The iteration can be described as a flow as seen in Figure 4.5. The classifier system iterates until the iteration number gets to the limit. When the classifier system stops, it outputs the classifiers for the user to analyze.

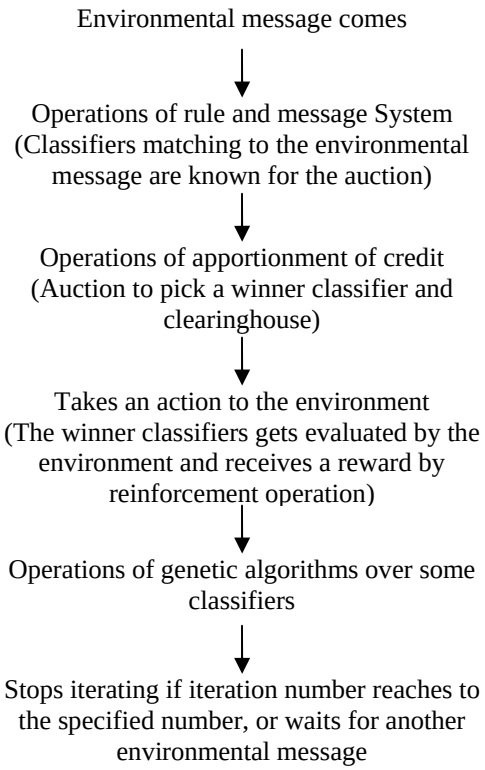

Figure 4.5: An Iteration of the Classifier System

4.4.1 Perform Class as the Rule and Message System

The rule and message system is often called performance system. Therefore the class is named *Perform*. The class diagram of *Perform* is given in Figure 4.6. The *Perform* has an operation *matchclassifiers* to find out who are matching to the environmental message. The parameter variable *emess* is the environmental message. For the matching classifiers, the value of the Boolean variable *matchflag* of *ClassType* is changed to *true* by this method. Among the matching classifiers, an auction takes place by the apportionment of credit system.

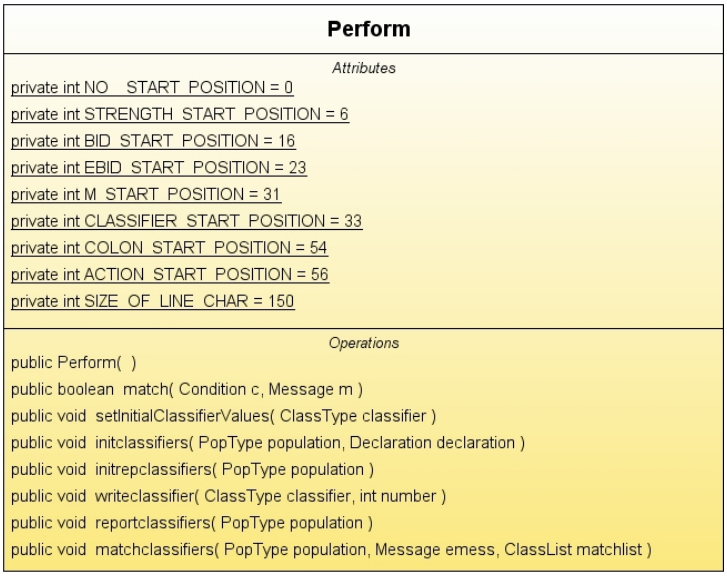

Figure 4.6: The Class Diagram of *Perform*

4.4.2 AOC Class as the Apportionment of Credit System

The apportionment of credit system is named *AOC* and Figure 4.7 shows its class diagram. The auction is done by the method *auction*. The returned integer value is the index of the winner classifier in population. The method *clearinghouse* method manages

the payment (by strength), and collects taxes. There are two kinds of taxes. One is bidding tax and the other is life tax. Therefore, bidder pays taxes even if it does not win. Life tax is an existence tax. Just being in the population generates a small fee. Some classifiers that bid but does not win at all simply lose the strength and become weak. Some classifiers that do not even bid lose the strength by the life tax.

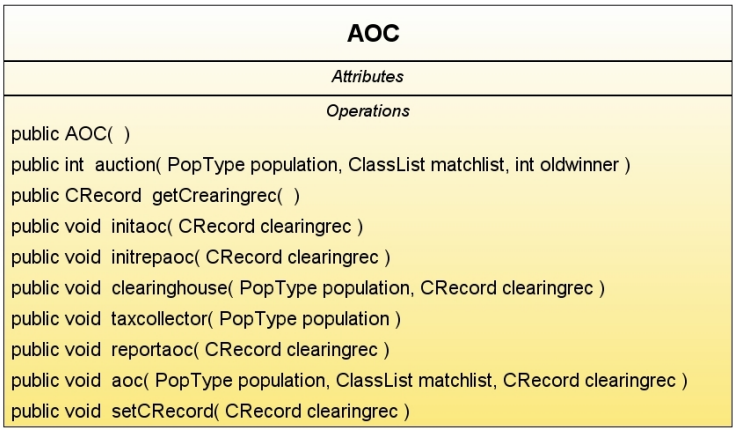

Figure 4.7: The Class Diagram of *AOC*

4.4.3 GA Class as the Genetic Algorithms

The genetic algorithms are implemented in the *GA* class as is in Figure 4.8. The *select* method is for the reproduction. The other two operators of genetic algorithms are implemented by the methods, *crossover* and *mutation*. The *select* method is simply the roulette wheel. The index of the picked classifier from the roulette wheel is returned by the method *select*. The *crossover* takes two parents picked by the *select* and generates the two children (offspring). The *mutation* is performed over the array of attributes given as the argument and returns the resulted attributes of the entity set.

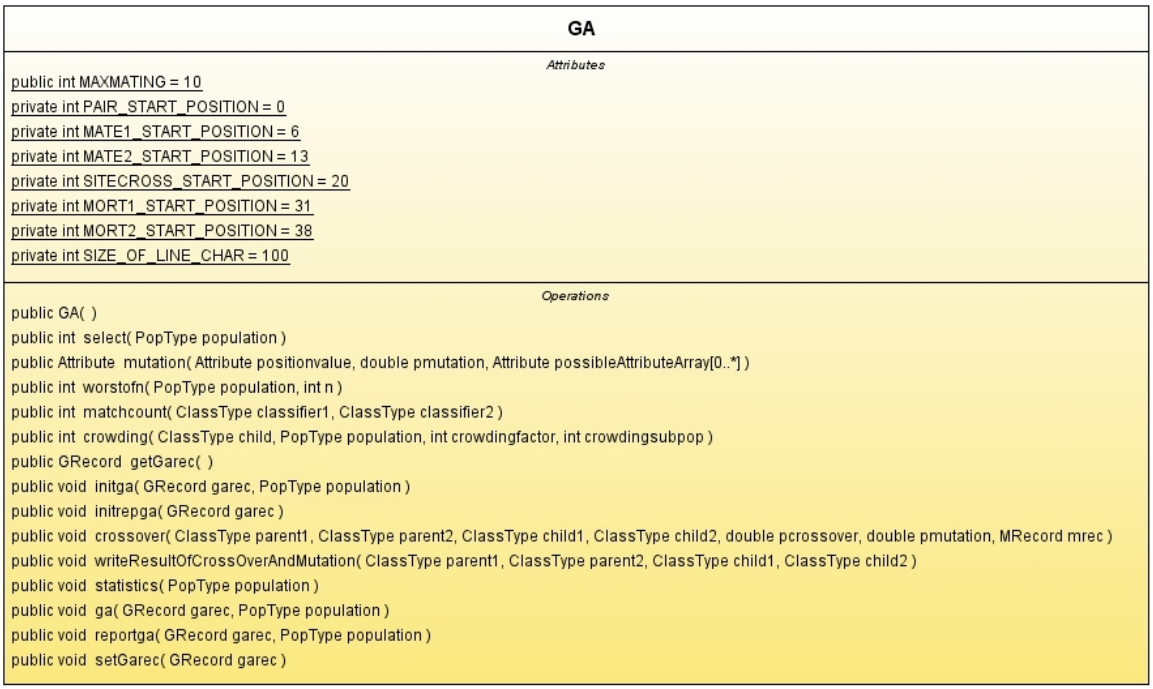

Figure 4.8: The Class Diagram of *GA*

4.4.4 Reinforc Class for the Reinforcement Learning

Figure 4.9 shows a class *Reinforc*, which implements the reinforcement learning. In every iteration, there is a winner classifier which receives a numerical reward value (strength). The winner can be thought to be the answer (action) from the classifier system to the environment. Therefore, the classifier system picks a classifier that the system thinks is the best. The auction process has the mechanism to perform the competition and to produce the winner, which is considered to be the best. Then the environment gives a reward to the winner classifier. The *Reinforc* implements such a mechanism. The method *payreward* gives a reward according to how good the winner classifier of a database schema is with the given knowledge of database design.

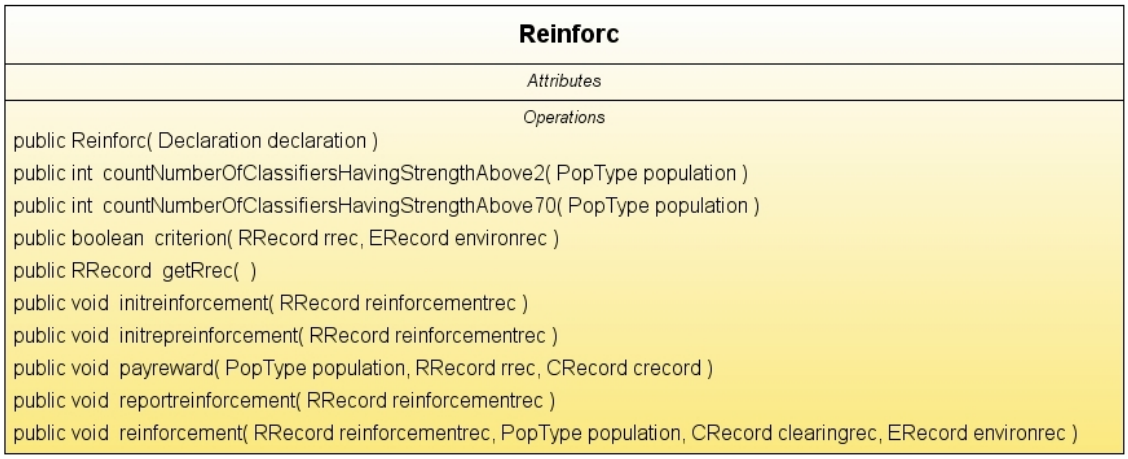

Figure 4.9: The Class Diagram of *Reinforc*

The *payreward* method uses a *RewardCalculator* class. The class diagram of *RewardCalculator* is given in Figure 4.10. The class has a set of functions that evaluate the winner classifier. Each function has a single rule or concept of database design and evaluates the winner. The reward value is the total value calculated by these functions and added to the strength of the winner classifier. When a new rule is enforced to classifiers, a new function is added to this class. Therefore, these functions navigate the learning, and the classifiers that are fitted to these rules or concepts get stronger. The following section shows one of the rules used in the classifier system.

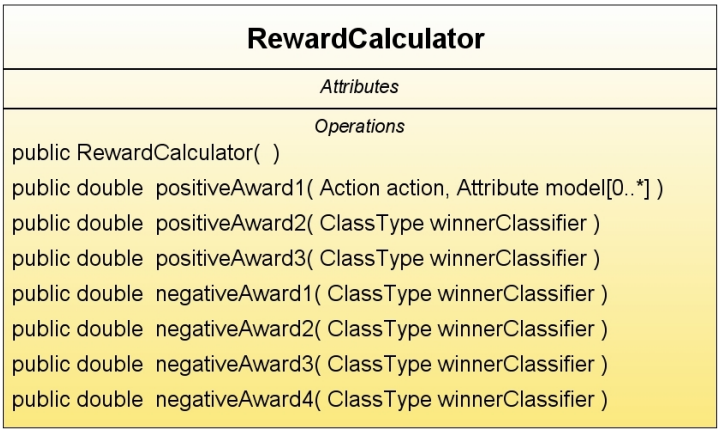

Figure 4.10: The Class Diagram of *RewardCalculator*

4.5 Knowledge given to the Classifier System & Reward Functions

Some database knowledge is given to the *Reinforc*. The classifiers represent schemas and there has to be a way to evaluate the winners. The evaluation depends on the knowledge. Whenever a new concept has to be added to the classifier system, another reward function is added to the *RewardCalculator* of *Reinforc*. Seven functions as in Figure 4.10 were used for experiment. Each has the rule to increase or decrease the strength of classifiers. For example, one of the rules is to decrease the reward value if the winner classifier contains a partial dependency. The classifier system is given some limited knowledge in the form of these functions, and the knowledge navigates classifiers to be fitted in the environment.

4.6 Example Operations over the Classifiers

In this section, some example processing of the classifiers by the four classes above are presented with some source codes. The most important operations are matching, auction, tax collection, genetic algorithms, and reinforcement. All of these operations

affect to the strength of classifiers and the learning result. This section shows how the classifiers are actually processed by those.

4.6.1 Operations of matchclassifiers in Perform

The source code of the method *matchclassifiers* is in Figure 4.11. The *matchlist* given at the parameter is used to maintain the list of index for matching classifiers. The method simply uses a loop to search the population and if the condition part of the classifier is matching to the environment message *emess*, then the value of *matchflag* is changed. Then, the index of the classifier is added to the *matchlist*.

```
public void matchclassifiers(PopType population, Message emess, ClassList matchlist) {
    matchlist.setNumOfActive(0);
    int nclassifier = population.getNclassifier();
    for (int j = 1; j <= nclassifier; j++) {
        ClassType classType = population.getClassifiers().getClassType(j);
        Condition c = classType.getC();
        classType.setMatchflag(match(c, emess));
        if (classType.isMatchflag()) {
            matchlist.setNumOfActive(matchlist.getNumOfActive() + 1);
            matchlist.setIndex(matchlist.getNumOfActive(), j);
        }
    }
}
```

Figure 4.11: The Method *matchclassifiers*

Figure 4.12 shows an example record that the classifier system maintains for single iteration. This contains the information of some classifiers in the population. The No. column indicates the index number of the classifier. Each of these classifiers is the candidate of database schemas for each entity set. The M column indicates if the classifier is matching to the current environmental message or not. If X is put in the column, then it indicates the classifier is matched. For the iteration, the classifiers with the index number from 8 to 10 are all matched. The index numbers of those are added to

the *matchlist.* An auction is held among the classifiers whose index numbers are in the *matchlist*.

No.	Strength	bid	ebid	M
1	0	0	0.15	
2	-0	-0	-0.09	
3	0	0	-0.12	
4	-0	-0	-0.1	
5	0	0	-0.01	
6	0	0	-0.05	
7	-0	-0	-0.04	
8	0	0	-0.05	X
9	0	0	-0.02	X
10	-0	-0	-0.02	X

Figure 4.12: Some Classifiers Matching to an Environmental Message

4.6.2 Operations of auction and taxcollector in AOC

Here, the source code of the *auction* method is provided in Figure 4.13. The *auction* method is given a *matchlist* as an argument that we saw in the method *matchclassifiers*. The auction sees each of the matching classifiers and finds the winner among them. The winner is the one who bid the most. The bidding value (*B*) is relative to the strength and calculated as follows:

$$B_i = C_{bid} S_i .$$

The C_{bid} is the bid coefficient, *S* is the strength and *i* is the classifier index. Therefore, the stronger classifiers can bid more than weaker ones. Actual comparison is done by the effective bid value. The effective bid (*EB*) value is calculated as follows:

$$EB_i = B_i + N(\sigma_{bid}) .$$

The N is a function of the specified bidding noise standard deviation σ_{bid}. The values of *cbid* and *ebid* in Figure 4.13 are the constant coefficients. The index of classifier whose effective bid is the highest is going to be returned by the function.

```
public int auction(PopType population, ClassList matchlist, int oldwinner) {
    double bidmaximum = 0.0; //max bid among the matching classifiers
    int k = -1;
    int winner = oldwinner; //if no match, old winner wins again
    int nactive = matchlist.getNumOfActive();
    if (nactive > 0) { //matching classifiers to environmental message
        for (int j = 1; j <= nactive; j++) {
            k = matchlist.getIndex(j);
            //with the classifier[k]
            ClassType classifier = population.getClassifiers().getClassType(k);
            double cbid = population.getCbid();
            double bid1 = population.getBid1();
            double bid2 = population.getBid2();
            double ebid1 = population.getEbid1();
            double ebid2 = population.getEbid2();
            double strength = classifier.getStrength();
            double bidsigma = population.getBidsigma();
            classifier.setBid(cbid * (bid1 + bid2) * strength);
            classifier.setEbid(cbid * (ebid1 + ebid2) * strength + DSDCS.utility.noise(0.0, bidsigma));
            if (classifier.getEbid() > bidmaximum) {
                winner = k;
                bidmaximum = classifier.getEbid();
            }
        }
    }
    return winner;
}
```

Figure 4.13: The Method *auction*

The source code of the method *taxcollector* method is provided in Figure 4.14. The *taxcollector* method collects bidding tax and life tax. The bidding tax is for classifiers who bid and the life tax is charged for every classifier for the iteration. The method sees the currently matching classifiers by seeing the value of the *matchflag*. Then matched classifiers are charged for the bidding tax. At the end of the method, life tax is subtracted from every classifier in the population.

```
public void taxcollector(PopType population) {
    double bidtaxswitch = 0.0;
    double lifetax = population.getLifetax();
    double bidtax = population.getBidtax();
    //life tax from everyone & bidtax from actives
    if (lifetax != 0.0 || population.getBidtax() != 0.0) {
        for (int j = 1; j <= population.getNclassifier(); j++) {
            ClassType classifier = population.getClassifiers().getClassType(j);
            if (classifier.isMatchflag()) {
                bidtaxswitch = 1.0;
            } else {
                bidtaxswitch = 0.0;
            }
            double strength = classifier.getStrength();
            classifier.setStrength(strength - lifetax * strength - bidtax * bidtaxswitch * strength);
        }
    }
}
```

Figure 4.14: The Method *taxcollector*

4.6.3 Operations of GA

The three operators of genetic algorithms are performed by the methods *select*, *crossover*, and *mutation*. Since the source codes are too long to list here, the example of the genetic algorithms over two classifiers is presented instead. Suppose there are 4 classifiers for EMPLOYEE as in Table 4.1. In the classifier system, usually there are hundreds or thousands of such classifiers for every entity set in the input. In Table 4.1, the No.1 and No.2 are seemingly good and No.3 and No.4 are weak.

No.	Condition	Message (Action)	Strength
1	Condition1	Message1	20
2	Condition2	Message2	20
3	Condition3	Message3	2
4	Condition4	Message4	0

Table 4.1: Four Classifiers for a Database Schema EMPLOYEE

The *select* method is likely to pick No.1 and No.2 to reproduce the offspring as a result of the roulette wheel. The strength indicates the measure of how good the database

schema candidates have been doing so far. The *crossover* is performed over the offspring classifiers, and the *mutation* happens rarely after the crossover operation. The crossover and mutation is performed on the message part of the offspring. These three methods of genetic algorithms sometimes change the content of classifiers and increase the performance of the classifier system.

4.6.4 Operations of payreward in Reinforc

A winner classifier is evaluated according to the reward functions of the *RewardCalculator* and receives a reward value from the *payreward* method. The input will affect the result of the *payreward* method. For example, the input conceptual schema may be given as follows:

EMP = {ENO, ENAME, TITLE}

ASG = {ENO, PNO, DUR, RESP}

PROJ = {PNO, PNAME, BUDGET}

PAY = {TITLE, SALARY}.

Here, EMP is for employee, ASG is for assignment, PROJ is for project, ENO is for employee number, PNO is for project number, and RESP is for responsibility. EMP is related to ASG and PAY. ASG is related to PROJ. All of them have one-to-one relationships. Figure 4.15 shows the relations and mapping cardinalities.

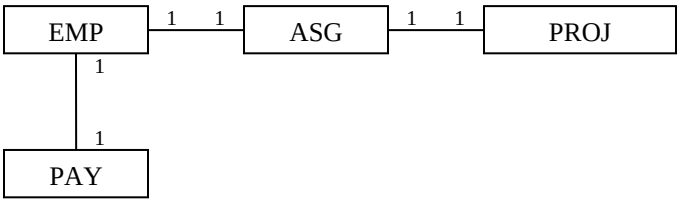

Figure 4.15: Relations and Mapping Cardinalities

`With this input information, the example winner database schema classifier is as follows:

ASG = {ENO, PNO, TITLE, SALARY}.

The input has some dependency information, given the primary keys. The system is given the mapping cardinality information between entity sets. Based on these input, together with the knowledge of reward functions, the system evaluate a winner database schema classifier. In the example winner, there is a partial dependency. Therefore, the classifier may be given a negative value. The mapping cardinality information may affect to the result, depending on the content of the classifier. The *Reinforc* are given seven functions and each is used to evaluate and give a numerical value to the winner as a reward.

Chapter 5: Result of an Experiment

This chapter shows an experiment with four database schemas as an input. The purpose is to see if the classifier system can learn good classifiers and bad classifiers of the database schema following the knowledge given to the reinforcement learning mechanism, rather than outputting completely well-designed or normalized database schema. The example database schema is the one used in Chapter 4 at Section 4.6.4. The schema is actually normalized to fifth normal form. Therefore, the input is a good example. The modified version of the database schema is also given to see the result when an input is badly designed. The system generates 2000 database schemas as classifiers and iterates 40000 times. Genetic algorithms are performed every 2000 iterations. Every time genetic algorithms take place, ten mates are selected from the population and the crossover is performed. The system was run ten times for each of the normalized and modified version of the database schemas.

5.1 Completely Well-Designed Database Schema as the Input

Figure 5.1 and 5.2 shows the input schema and the mapping cardinalities. Figure 5.3 shows 1 percent of the initial population members among 2000 classifiers generated by the system. All the classifiers of the initial population are evaluated by the reward functions before the first iteration starts. Thus the ranking starts just after they are created. As seen in Figure 5.3, some are strong and some are weak initially. These 2000 classifiers will be in auctions, sometimes changed by genetic algorithms, and ranked by reinforcement learning mechanism.

No.	Entity	PK Attribute	Possible Non-PK Attribute
1	EMP	ENO	ENAME TITLE
2	PROJ	PNO	PNAME BUDGET
3	ASG	ENO PNO	RESP DUR
4	PAY	TITLE	SALARY

Figure 5.1: Well-Designed Input Schema

No.	Left Entity set	Mapping cardinality	Right Entity set
1	EMP	1-1	ASG
2	ASG	1-1	PROJ
3	EMP	1-1	PAY

Figure 5.2: Mapping Cardinalities of the Well-Designed Input Schema

No.	Strength	bid	ebid	M
1	-1.33	0	0	
2	3	0	0	
3	1	0	0	
4	3	0	0	
5	0.67	0	0	
6	1.33	0	0	
7	0.67	0	0	
8	1	0	0	
9	2	0	0	
10	3.33	0	0	
11	1.33	0	0	
12	0.67	0	0	
13	1.5	0	0	
14	0.67	0	0	
15	3	0	0	
16	1	0	0	
17	5.33	0	0	
18	1.33	0	0	
19	2	0	0	
20	-1.5	0	0	

Figure 5.3: Initial Population Generated From the Well-Designed Input Schemas

The classifiers experience the environment until the system iterates 40000 times. Figure 5.4 shows the top five of the final population sorted by strength. The classifiers

from No.1 to No.4 were the strongest ones. For the run, the strongest classifiers were exactly the same as the well-designed database schemas, and the classifier system successfully increased the strength of good classifiers. All of the other classifiers had strength below 0.39.

No.	Strength	bid	ebid	M
1	54.55	5.45	5.58	
2	54.55	5.45	5.5	X
3	54.55	5.45	5.39	
4	27.27	2.73	2.69	
5	0.39	0.04	-0.01	X

Figure 5.4: Top Five Classifiers of the Final Population in the Experiment with the Well-Designed Schemas

Figure 5.5 and 5.6 are the plots for the number of classifiers having strength over 2 and over 10, respectively. As seen in Figure 5.5, there were initially over 600 classifiers having strength over 2. However, the number changes as the system iterates. Some particular classifiers increase the strength and some others decrease the strength. In the plot of Figure 5.6, there are sudden increases of classifiers with the strength over 10, for every 2000 iteration interval. This is because of the genetic algorithms operation. Finally, the number of classifiers with strength 10 converges to the number of entity sets. Out of 10 runs of the system, the similar plots were always obtained. The strongest classifiers were the same or almost the same as the well-designed input database schemas.

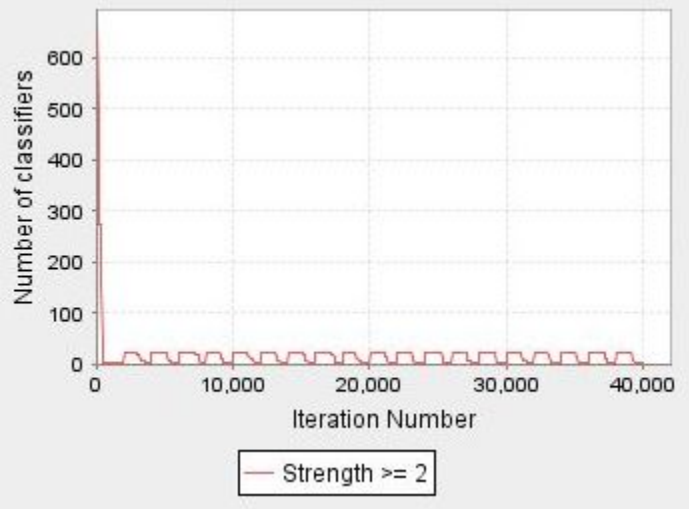

Figure 5.5: Number of Classifiers with Strength Over 2

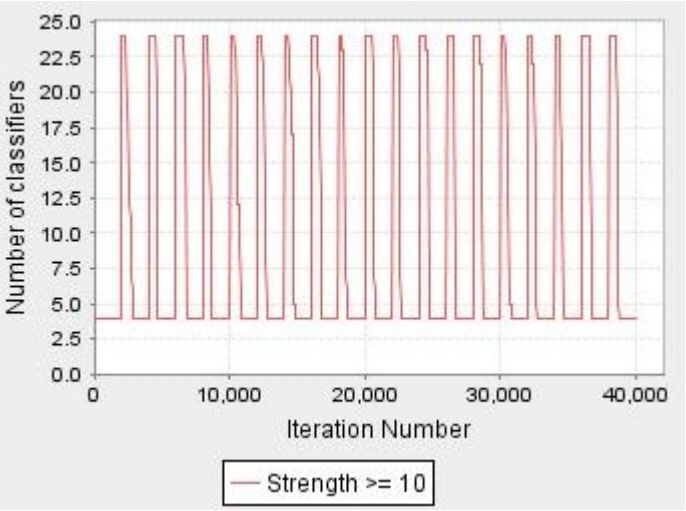

Figure 5.6: Number of Classifiers with Strength Over 10

5.2 Badly Designed Database Schema as the Input

This time, one of the input schemas is not normalized. The mapping cardinality is the same as before. The input schemas change as Figure 5.7. Here, the non-primary key attributes of PROJ is changed. The modified PROJ schema violates third normal form,

since there is a partial dependency, TITLE → SALARY. The TITLE is a primary key of PAY, and PROJ is not related to PAY. The classifier system is given some knowledge that may detect this. Figure 5.8 shows some of the classifiers in the initial population. In the initial population of 2000 classifiers, there were many candidate classifiers of PROJ schema with the partial dependency.

No.	Entity	PK Attribute	Possible Non-PK Attribute
1	EMP	ENO	ENAME TITLE
2	PROJ	PNO	PNAME BUDGET TITLE SALARY
3	ASG	ENO PNO	RESP DUR
4	PAY	TITLE	SALARY

Figure 5.7: Input in Which PROJ Is Modified

No.	Strength	bid	ebid	M
1	2	0	0	
2	-1.67	0	0	
3	-2.33	0	0	
4	-1.67	0	0	
5	0.33	0	0	
6	2.67	0	0	
7	-2	0	0	
8	3.2	0	0	
9	4.8	0	0	
10	-0.6	0	0	
11	2	0	0	
12	2.6	0	0	
13	0.5	0	0	
14	2.2	0	0	
15	-0.5	0	0	
16	3	0	0	
17	0.4	0	0	
18	-1.8	0	0	
19	2	0	0	
20	3	0	0	

Figure 5.8: Initial Population Generated From the Badly Designed Input Schemas

Figure 5.9 shows the top five classifiers sorted by strength in the final population. The first four includes each entity set and the PROJ did not have TITLE and SALARY attributes. The bad classifiers for PROJ decreased the strength and the strongest one increased the strength as the system iterates, and thus the classifier system successfully eliminated the bad schema. Out of ten runs, the system always excluded the partial dependency.

No.	Strength	bid	ebid	M
1	54.55	5.45	5.4	
2	45.45	4.55	4.58	
3	43.64	4.36	4.41	
4	27.27	2.73	2.91	X
5	0.53	0.05	0.16	

Figure 5.9: Top Five Classifiers of the Final Population in the Experiment with Badly Designed Schemas

Chapter 6: Conclusion

This thesis presented the implementation of the classifier system which learns good and bad database schemas. The result of experiment showed that the system has an ability to choose the good database schema, by learning. The system iterated 40000 times and good classifier increased and bad classifier decreased the strength. At the end of the iterations, there were exclusively strong classifiers for each entity set. The candidate classifiers for each entity set converged to the one which is good according to the knowledge.

References

[1] Siberschatz, A., Korth, F. H., Sudarshan, S., “Database System Concepts”, McGraw-Hill, 5th edition, 2006.

[2] Chen, P. P., “The Entity-Relationship Model-Toward a Unified View of Data”, ACM Transactions on Database Systems (TODS), Volume 1, Issue 1, pp. 9 – 36, 1976.

[3] Codd, F. E., “A Relational Model of Data for Large Shared Data Banks”, Communications of the ACM, Volume 26, Issue 1, pp. 64 – 69, January 1983.

[4] Özsu. M. T., Valduriez, P., “Principles of Distributed Database Systems”, Prentice-Hall, 2nd Edition, 1999.

[5] Gary, R. M., Johnson, S. D., “Computers and Intractability A Guide to the Theory of NP-Completeness”, W. H. Freeman and Company, 1979.

[6] Buchanan, G. B., “A (Very) Brief History of Artificial Intelligence”, AI Magazine, Volume 26, Issue 4, pp. 53 – 60, Winter 2005.

[7] McCarthy, J., “What is Artificial Intelligence?”, http://www-formal.stanford.edu/jmc/whatisai/whatisai.html.

[8] Campbell, S. M., “Knowledge Discovery in Deep Blue”, Communications of the ACM, Volume 42, Issue 11, pp. 65 – 67, November 1999.

[9] Seirawan, S., Simon, A. H., Munakata, T., “The Implications of Kasparov vs. Deep Blue”, Communications of the ACM, volume 40, Issue 8, pp. 21-25, August 1997.

[10] Mitchell, T. M., “Machine Learning”, WCB McGraw-Hill, 1997.

[11] Goldberg, E. D., “Genetic Algorithms in Search, Optimization & Machine Learning”, Addison-Wesley, 1989.

[12] Goldberg, E. D., “Dynamic System Control Using Rule Learning and Genetic Algorithms”, Proceedings of the 1st International Conference on Genetic Algorithms, pp.8 – 15, 1985.

[13] Wilson, S. W., “Classifier System Learning of a Boolean Function”, Research Memo RIS No. 27r, The Rowland Institute for Science, Cambridge, MA.

[14] Holland, J. H., “Outline for a Logical Theory of Adaptive Systems”, Journal of the ACM (JACM), Volume 9, Issue 3, pp. 297 – 314, 1962.

[15] Darwin, C. R. 1859. On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life. London: John Murray. 1st edition, 1st issue.

[16] Holland, J. H., “Adaptation in Natural and Artificial Systems”, The MIT Press, 1992.

[17] Kellerer, H., Pferschy, U., Pisinger, D., “Knapsack Problems”, Springer, 2004.

[18] Smith, R. E., Goldberg, E. D., “Reinforcement Learning With Classifier Systems: Adaptive Default Hierarchy Formation”, Applied Artificial Intelligence, Volume 6, Issue 1, pp. 79 – 102, 1992.

[19] Davis, R., King, J., “An Overview of Production Systems”, In E. W. Elcock & D. Michie (Eds.), Machine Intelligence 8, pp.300-332, Wiley, 1976.

Appendices

Appendix A: Permissions

Div 542: Code 9780201157673
Req No. 31866: Cust No.13708

Mitsuru Tanaka
3900 Hessmer
Metairie, LA 70002

is hereby granted permission to use the material indicated in the following acknowledgement. This acknowledgement must be carried on the copyright or acknowledgements page of your thesis and as a footnote of the pages that contains our material:

David E. Goldberg, GENETIC ALGORITHMS IN SEARCH, OPTIMIZATION, AND MACHINE LEARNING, (include the page number from our book that the material has been taken from) © 1989 Pearson Education Inc. Reproduced by permission of Pearson Education, Inc.

This material may only be used in the following manner:

To reprint and include tables/figures, detailed below, in your thesis "Classifier System Learning of Good Database Schema". This thesis will be presented to the University of New Orleans in one print and one electronic copy. The thesis will be available at http://louisdl.louislibraries.org/index.php?name=university%20of%20New%20Orleans%20Electronic%20Theses%20and%20Dissertations.
If you should wish to have your thesis professionally published for the commercial market you must re-apply for permissions.
"Genetic Algorithms" page one, "Robustness" page one, "Search Procedures" page 7, "Classifier Systems" page 221, "Schema theorem" page 30-33, "figures 6.1 from page 223 and 6.3 from page 227, tables 1.1 from page 11, and 1.2 from page 16, "A Simple Genetic Algorithm" form page10, "A Simple
Classifier Systems by Hand" page 227.

This permission is non-exclusive and applies solely to the following language(s) and territory:

Language(s): English
Territory: Thesis

IMPORTANT NOTICE:
Pearson Education, Inc. reserves the right to take any appropriate action if you have used our intellectual property in violation of any of the requirements set forth in this permissions letter. Such action may include, but is not limited to, the right to demand that you cease and desist in your use of Pearson Education's material and remove it from the marketplace.

This permission does not allow the reproduction of any material copyrighted in or credited to the name of any person or entity other than the publisher named above. The publisher named above disclaims all liability in connection with your use of such material without proper consent.

Tim Nicholls, Manager Rights and Permissions
Pearson Education
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116

Vita

Mitsuru Tanaka was born in Matsumoto city of Nagano Prefecture in Japan. He received the degree of Bachelor of Engineering from Hiroshima Kokusai Gakuin University in March, 2001. In January, 2005, He entered the Graduate School at the University of New Orleans. Currently, he is working as a research assistant and completing the degree of Master of Science in Computer Science at the University of New Orleans.